Preprint

A Data Pump for Communication

Myong H. Kang and *Ira S. Moskowitz*

FROM:
Submitted for publication 1994/1995.

CONTACT:
Myong H. Kang, Information Technology Division, Center for High Assurance Computer Systems, Mail Code 5542, Naval Research Laboratory, Washington, DC 20375.

Ira S. Moskowitz, Information Technology Division, Center for High Assurance Computer Systems, Mail Code 5543, Naval Research Laboratory, Washington, DC 20375.

E-MAIL:
mkang@itd.nrl.navy.mil
moskowit@itd.nrl.navy.mil

COMMENTS:(version — May, 16 1994)
Corrected typos in some references

A Data Pump for Communication*

Myong H. Kang and Ira S. Moskowitz
Information Technology Division — CHACS: Code 5540
NAVAL RESEARCH LABORATORY
Washington, D.C. 20375

Abstract

As computer systems become more open and interconnected, the need for reliable and secure communication also increases. In this paper, we introduce a communication device, the Pump, that balances the requirements of reliability and security. The Pump provides acknowledgements (ACKs) to the message source to insure reliability. These ACKs are also used to regulate the source to prevent the Pump's buffer from becoming/staying full. This is desirable because once the buffer is filled there exists a huge covert communication channel. The Pump controls the input rate from the source by attempting to slave the input rate to the service rate through the randomized ACK back to the source.

An analysis of the covert channel is also presented. The purpose of the covert channel analysis is to provide guidelines for the designer of the Pump to choose appropriate design parameters (e.g., size of buffer) dependent upon the analysis presented in this paper and system requirements.

1 Motivation

Sharing information between users/processes, or, more simply put, *entities*, is undeniably the wave of the future. This will be true, regardless of whether this sharing is centralized on one computer or, at the other extreme, distributed over the information highway. As computer systems become more open and distributed, the security concerns relating to information exchange between different entities will grow.

Two security concerns are paramount:

*Parts of this paper have previously appeared in "A Pump for Rapid, Reliable, Secure Communication", 1st ACM Conference on Computer & Communications Security '93-11/93, pages 119-129, ACM Press.

1. There should be no intrusion of unauthorized entities. Research on access control, virus detection and prevention, etc. focus on this aspect of security.
2. No information should flow to unauthorized entities in the computer systems.

In this paper, we are looking at the second concern—security means no unauthorized information flows. We assume all entities are authorized to be in/on the system; however these entities may still act in a malicious manner. We do not want to have certain entities know what is in certain files, i.e., we do not want these entities to be able to read these files and we do not want entities that *are* authorized to read these files to send any data to the unauthorized entities. What we are discussing is a multilevel secure system where different entities have different sensitivity levels that form a lattice [1, 2]. The prohibitions against "reading up" and "writing down" described above are the Bell-LaPadula requirements (BLP) [3, 2]. In an information theoretic sense this is equivalent to requiring that information only flow from a lower level entity (Low) to a higher level entity (High).

At first glance, there seems to be no problem meeting the BLP requirements. Simply design a system so that information only flows up, not down. Let us examine a conventional (non-secure) approach to sending information from one entity to another and see what problems occur. Throughout this paper, we assume that there is only one source entity and only one destination entity. An application of the Pump in secure networks, using the foundations developed here, is dealt with in a separate paper[4].

2 The Store and Forward Protocol—SAFP

A conventional communication protocol used in (non-secure) systems should include the following characteristics:

1. *Reliability*: The communication is reliable if there is no loss of messages and no duplication of messages.
2. *Good Performance*.

A standard conventional protocol is the SAFP.

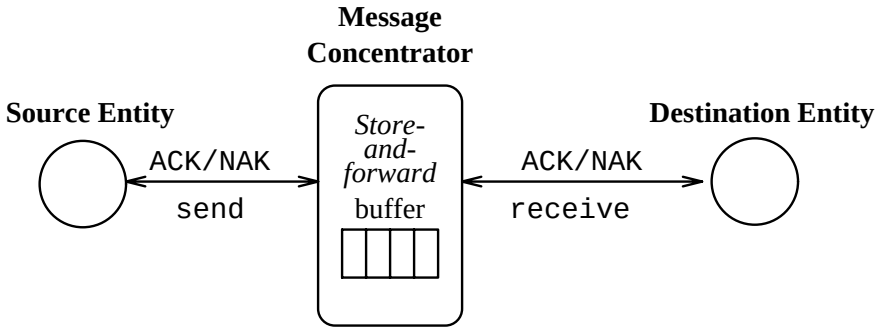

Figure 1: Message passing from source entity to destination entity under the SAFP.

Figure 1 shows a typical example of the SAFP. If the message-passing occurs in a distributed environment, then the source and destination entities may reside in two different computers and the message concentrator itself may be yet another computer; if the message-passing occurs in a single computer, then the operating system may play the role of the message concentrator (e.g., *pipes* in a UNIX system).

A typical message-passing scenario, in the SAFP, between the source entity (message concentrator) and the message concentrator (destination entity) goes as follows:

1. Establish transmission/connection.
2. Send a message.
   - If the source entity (message concentrator) receives an ACK, then discard the message from the source entity's memory (message concentrator's buffer).
   - If the source entity (message concentrator) either receives a NAK or times out, then retransmit the message.
3. If there are more messages to send, then go to step (2).
4. Signoff/Disconnection.

Note that step 2 above guarantees reliability because if a NAK is sent or if the message is timed out, then the source entity retransmits. Also, the ACKs allow the source entity to perform garbage collection. By this we mean that the source entity can reclaim memory (buffer space) and reuse it later. If a non-volatile buffer is used, the above communication is recoverable from system (except media) failure. We assume that message identification numbers are used so that duplicate messages can be easily handled.

If the source entity sends messages faster than the destination entity can receive them (either due to slow processing or failure in reception), then the buffer in the message concentrator may be filled. The source entity then will be unable to send any more messages until the destination empties some messages from the message concentrator. In this case, we say that the source has been *blocked*. However, if the source has a slower rate than the destination, determining the size of buffer that keeps the message blocking probability within specified design limits has been widely studied [5, 6], and closed form, solutions have been obtained provided we assume a M/M/1 queue.

In a secure environment, if Low is the source entity and sends messages to High (destination entity), then the same protocol cannot be used. Since the time at which Low receives the ACK/NAK may be under the control of High the ACK/NAK arrival times can be used to send information from High to Low (we discuss this fully in section 3.2.). If we do away with the ACK/NAK protocol, we can insure security but at the price of unacceptable degradation of reliability, recoverability, and/or performance — these are the problems with the read-down and blind write-up protocols (which do not violate BLP) that we will discuss next. Therefore, we modify the SAFP to allow Low to pass messages to High in a secure manner. For the rest of this paper Low will be the source entity and High will be the destination entity.

2.1 Read-Down Protocol

Read-down allows High to read Low's memory. But it does not allow High to send acknowledgements after it reads Low's message; hence, this protocol is secure. Consider the following implementation of a mechanism that passes information from Low to High using only *read-down*. Low inserts its message into a low level message buffer (in figure 2 the dashed line separates security levels). High reads the message, but Low has no indication that the message has been read. The message sits unchanged in the buffer until Low deletes it.

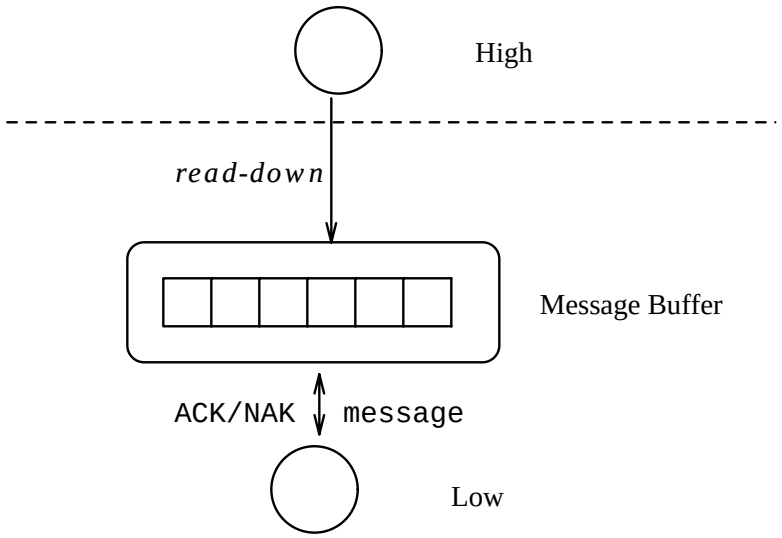

Figure 2: Message passing from Low to High using read-down.

Assuming that no error has occurred in the read-down procedure, two ways to achieve this communication are:

- *High continuously polls the low buffer.* However, this method wastes resources (e.g., CPU time)—hence, a performance penalty.
- *High periodically performs a read-down (e.g., every Δ time).* In this case, Low cannot send more than one message per Δ time units. Otherwise, (unless there is an infinite buffer) Low may delete messages which have not been read by High. If Δ is too small, then, like the polling method, this method will waste resources. If Δ is too large, then the message rate will be reduced (i.e., the message rate of this communication is less than or equal to $\frac{1}{\Delta}$ *messages/unit time*). Hence, a performance penalty.

Another drawback of this method is that Low cannot detect if High is ready to receive messages or not. For example, if High crashes, there is no way for Low to detect the situation and stop sending messages (otherwise Low may delete messages that High has not read yet).

2.2 Blind Write-Up Protocol

Blind write-up allows Low to write on High's memory. But it does not allow High to send an ACK/NAK to Low. We could implement a blind write-up mechanism as follows in figure 3.

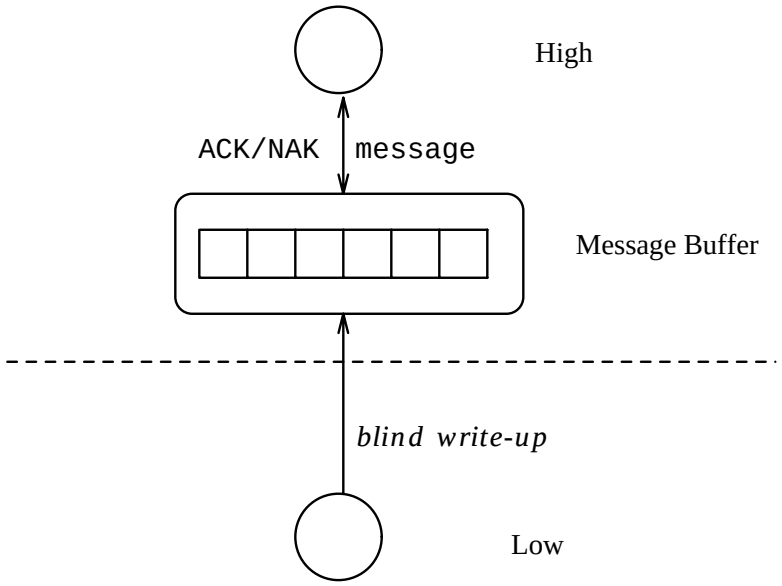

Figure 3: Message passing from Low to High using *blind write-up*.

Low writes its message into the high level message buffer and High reads messages from the buffer. Low does not know whether or not the message buffer has space available (since this message buffer is at higher level than Low), so it must send a message and hope that High receives it. Hence, this mechanism is unreliable because even if an error occurred during transmission, there is no way for Low to discover it and retransmit the message. Also, Low might write over messages before High has read them.

3 A Quasi-Secure Low-to-High Communication Protocol

The communication mechanisms presented in the previous section all have undesirable characteristics. The read-down and blind write-up methods are unreliable because there is no way of knowing if the intended receiver actually received the message. Hence, even though these mechanisms may be secure, they are not a good substitute for the conventional protocol. We believe that there is no way to obtain a realistic system with realistic performance requirements without sacrificing some security. Therefore, we seek to design a system with **quasi-security** that still satisfies the performance and reliability requirements. In other words, we will tolerate a system that allows some insecurity below a certain level provided that the rest of the requirements are satisfied. To quantify that level, we must discuss covert channels. Classically,

a covert channel has been defined as a communication channel from High to Low that exists contrary to a secure system design [7]. Since covert channels seem to be a fact of life, we adopt the more lenient view and just say that in a MLS system any communication channel from High to Low is a covert channel. We say that the system is quasi-secure if the potential damage of a covert channel is kept within tolerable bounds. Capacity (in the information theoretic sense of Shannon [8]) of the covert channel is a good measure of insecurity if we are concerned with the leakage of a large amount of data in a large amount of time. If our concern is with a small message being passed covertly, possibly in a very noisy environment, then we need another metric aside from capacity (this is discussed fully in the section on the *Small Message Criterion* in [9]). Hence quasi-security must take the small message criterion into account also. In this paper, we can think of the small message criterion as a bound for how many bits can be passed (possibly noiselessly) in a small amount of time.

There are two basic types of covert channels [10]; the storage channel (different responses, same time) and the timing channel. A covert channel is called a *timing channel* if the output alphabet consists of the same response given at different times. Our security concern with the conventional communication protocol is that of High forcing Low ACKs to arrive at different times and thus creating a timing channel.

The **Pump**, introduced in this section, is a variation of the conventional communication protocol that was introduced in section 2. We already mentioned that the conventional communication protocol has a covert channel. One way to circumvent this timing channel problem is to limit the ACK/NAK sending rate to meet the NCSC covert channel capacity guidelines [11] for B3/A1 systems. However, if it is desirable to send more messages (or ACK/NAK) than what the NCSC guideline specifies, and the communication channel can handle this traffic, then this limitation severely penalizes the performance of the communication system.

The Pump adds random noise to conventional communication methods to reduce the covert channel capacity. There have been other attempts to reduce channel capacity by introducing random noise to the system [12, 13, 14, 9]. Our approach is different in the sense that ours pays almost no performance penalty in the benign situation (i.e., there is no Trojan horse in the system). When Trojan horses attempt covert communication our approach reduces potential timing channel capacity. Before describing the Pump in detail, we first examine the timing channel that exists in the conventional communication protocol.

3.1 The Full Buffer Channel

We are looking at a secure system where, as stated before, the source entity is Low and the destination entity is High. Below, as before, is the conventional communication protocol illustrated for such a secure system.

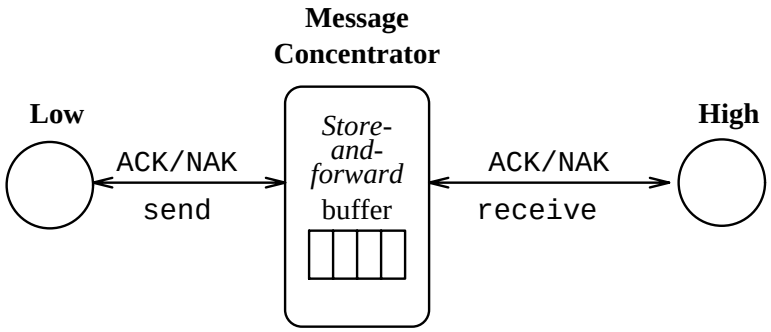

Figure 4: Conventional message passing from Low to High.

High cannot directly communicate with Low. However, Low must receive an ACK before it sends its next message to the (store and forward) buffer. High, by not sending ACKs to the buffer, can cause the buffer to become full. As long as the buffer is not full Low receives the ACK after a wait of overhead communication time O_v (For the sake of simplicity we assume that the Low and High ACK overheads are the same. If the overheads differ our formulas in section 3 can be easily modified). Once the buffer becomes full, High can send an ACK to the buffer, a message is removed from the buffer, a space is open on the buffer, and Low will receive an ACK. The time at which this ACK arrives at Low is under the direct control of High. Note that when Low receives its ACK, the buffer becomes full again and High can repeat the game. Thus we see that there is a covert timing channel between High and Low which we refer to as the full buffer channel (FBC).

3.1.1 Trojan horse Exploitation of the FBC

A Trojan horse (malicious software in both Low and High) can exploit the present situation and create a covert timing channel. The Trojan horse controls when Low sends a message and controls when High sends an ACK back to the message concentrator.

- The Trojan horse fills the buffer by not removing messages from the SAFB. Now that the buffer is full, a noiseless covert timing channel exists between Low and High. Furthermore, this noiseless channel exists as long as the buffer is full.
- Now Low sends a message to the SAFB. The SAFB cannot send an ACK back to Low until a spot opens up on the buffer. If High removes a message as soon as (or before) Low sends a message than Low only waits the overhead time O_v for an ACK. We assume that High, by removing messages from a full buffer, can affect the ACK time to Low in increments of $i\epsilon, i = 0, 1, 2, \ldots$. Since the high Trojan horse knows the size of the buffer (i.e., n) and how fast the low Trojan horse can send a message, High knows that Low has filled the buffer and has just sent a new message to the buffer. If Low gets an ACK at time $O_v + i\epsilon$, Low interprets the signal as the $(i+1)$st symbol. Since every time Low receives an ACK, the buffer is full again, and Low can then attempt to insert its new message — High can noiselessly send symbols again.

With this example we are looking at a worst case scenario. High will try to send symbols as quickly as possible, hence the time values of $O_v + i\epsilon$. Of course this allows unbounded response times. Real systems have timeouts and we assume that the timeouts in question are very large in comparison to $O_v \geq \epsilon$, so that by examining a channel with i bounded, changes the capacity very little from assuming that i is unbounded. The time units of our system are such that ϵ is an integer, i.e., ϵ is an integer number of system clock ticks. The channel capacity [8] of this channel is given by

$$C = \limsup_{k \to \infty} \frac{\log N(k)}{k} \text{ bits per clock tick}$$

where the logarithms are base two and $N(k)$ is the number of distinct sequences of symbols (ACK times) that take a total of time k. It can be shown [15, 16] that $C = \log \omega$, where ω is the positive root of $1 - (x^{-O_v} + x^{-\epsilon})$. The polynomial arises from the recurrence relation $N(k) = N(k - O_v) + N(k - (O_v + \epsilon)) + N(k - (O_v + 2\epsilon)) + \cdots$.

Define q by $\frac{O_v}{\epsilon} = q$. Note that q need not be an integer. By changing variables and letting $y = x^{\epsilon}$ we see that ω^{ϵ} is the positive root of $1 - (y^{-q} + y^{-1})$. Unfortunately, the only closed form solution for such polynomials [15] involve special functions. We will give the following figure to illustrate the various capacities. (Note for certain special cases such as $q = 1$ we can obtain the trivial closed form solution that $C = 1/\epsilon$ bits per clock tick. Similarly for $q = 2$, we have $C = \epsilon^{-1} \log \frac{1+\sqrt{5}}{2}$.)

Below is a plot of ϵC, as a function of q

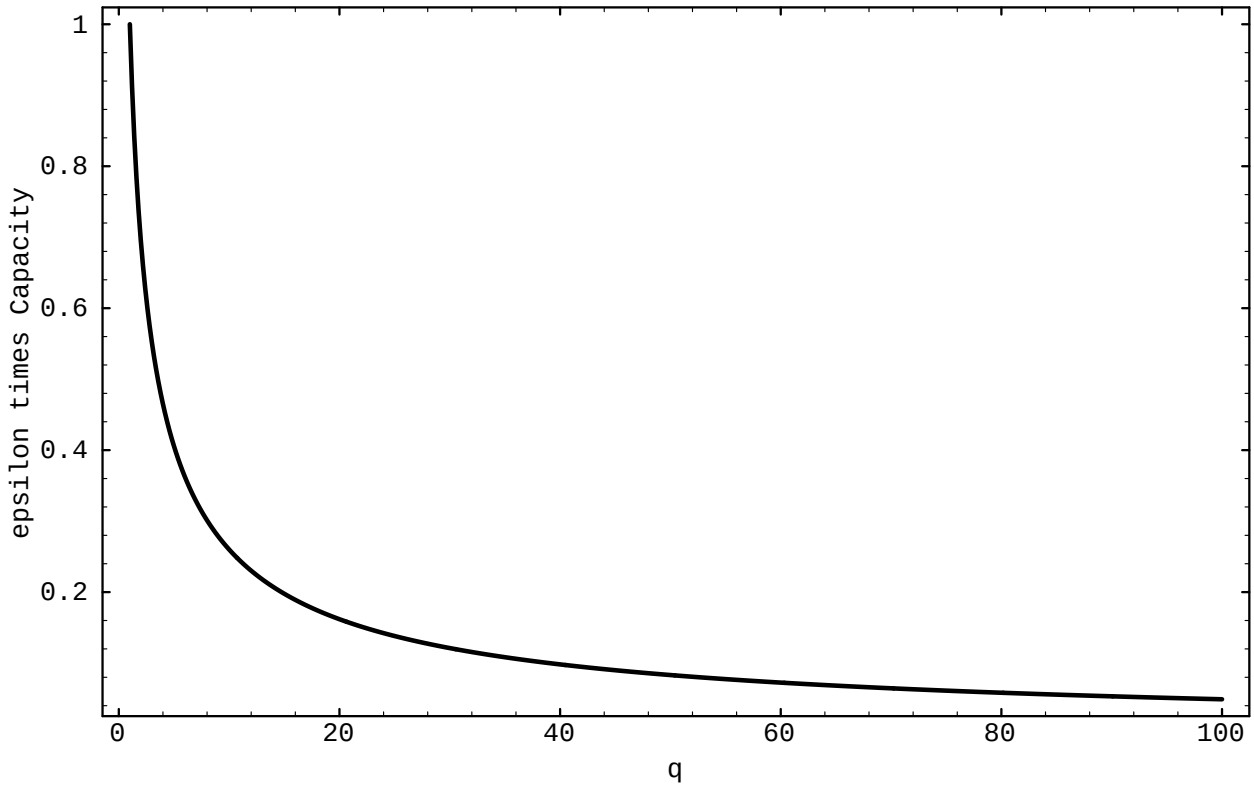

Figure 4.a: ϵC of the FBC.

No matter how large the buffer may be, the important fact is that eventually the buffer will fill up. (The Pump does not allow the full buffer channel to exist because the Pump keeps the buffer from being/remaining full.)

3.1.2 The FBC and relative Low/High speeds

We need to consider the relative speeds or rates of the Low and High processes. The rates are the inverse of the mean time for arrival or service. The rate that Low can send messages to the buffer, provided the buffer is not full, will be called the *arrival rate* and the rate at which High ACKs messages from the buffer will be called the *service rate*. The actual arrivals, service times, ACK times, etc. are often governed by probabilistic distributions. If the arrival rate is slower than, or equal to, the service rate then the buffer will not stay full and we do not have the FBC (the buffer can temporarily become full through bursty Low behaviour). However,

if the arrival rate is faster than the service rate, or if a Trojan horse is present to slow High down, (thus Low would then be faster) the buffer can stay full. Therefore, to prevent the FBC we need to slow the arrival rate down. We can accomplish this by slowing the Low ACK rate (R_l) down. Since Low cannot send a new message until it has an ACK from its previous message, we see that the slowing of R_l rate also slows the arrival rate (i.e., back-pressure). Of course we can slow R_l down as much as we like but for security purposes we need only slow R_l down to match the service rate. Any reduction past this point degrades performance. Note that service rate is the same as the High ACK rate (R_h), whereas the arrival rate is quite different from R_l. These distinctions are quite important when we discuss the queuing theoretic simulations.

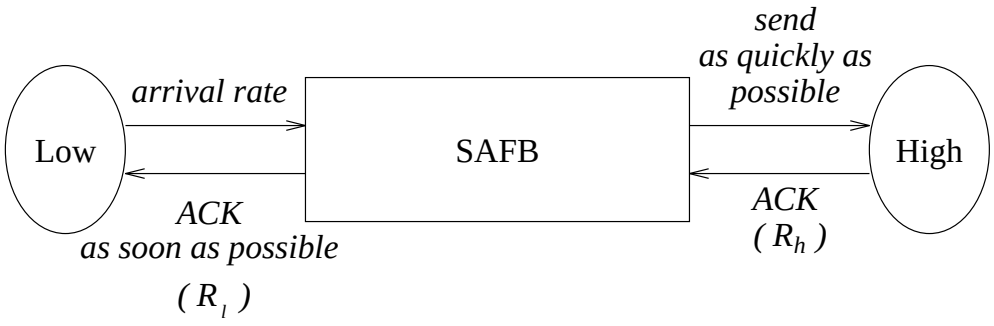

Figure 5: Rates associated with the conventional communication protocol.

What the Pump does is to slow the arrival rate down to match R_h if the arrival rate starts out faster than the service rate. The Pump lowers R_l, thus lowering the arrival rate, by basing R_l on a moving average of past High ACK times. If the arrival rate is slower than R_h the Pump basically leaves the arrival rate unchanged.

3.2 The Pump—A Quasi-Secure Communication Protocol

This process can be used as a communication mediator between any two security levels. Even though the Pump can reside in either the low or high level, in this paper we assume that the Pump resides in the security level of the destination entity and hence is of a high level. The Pump needs to be "trusted" in the sense that the system designer has an assurance that the Pump will do only what it is supposed to do (i.e., the Pump sends to Low only ACK/NAK and does not repeat High's message). In a sense, the Pump is blocking any message flow from High to Low.

In our model of communication between Low and High, the location (i.e., either in the same computer or in two separate computers) of these two processes is not important.

The Pump has three basic components which work in conjunction with (Pump independent) Low and High to allow data to be passed from Low to High. In actuality, there is a subtle violation of Bell-LaPadula which allows covert channels, of small capacity, to exist.

This is why we call the Pump quasi-secure. The "trust" insures that there are no further Bell-LaPadula violations. We will examine this later in the paper.

The components are the the trusted low process (TLP), the trusted high process (THP), and a communication buffer (CB). The Pump works as follows:

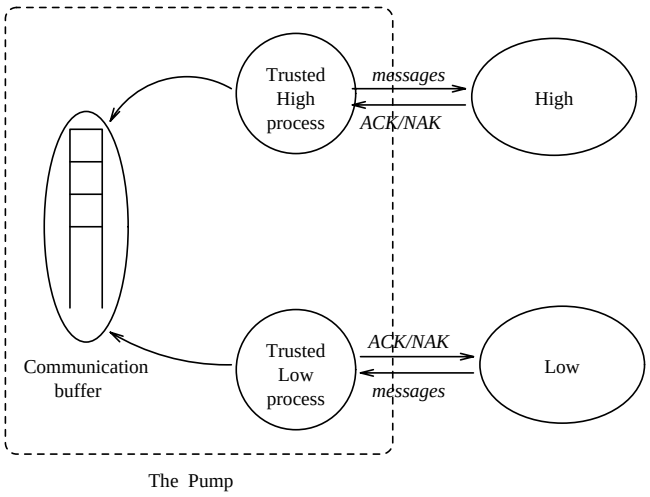

Figure 6: Message passing from Low to High using the *Pump*.

Low: (Exterior to the Pump)

Low sends a message to the TLP and waits for an ACK from the TLP. Once an ACK arrives, then the message is removed from Low (i.e., does the garbage collection from Low's internal queue) and a new message is sent (i.e., if Low receives NAK or no response due to a time-out then it will retransmit the same message). Note that Low may prepare a new message while Low waits for an ACK/NAK[1].

Trusted low process:

When a message arrives from Low, the TLP inserts the message in the CB and then sends an ACK, after a certain probabilistic delay based on a moving average of the past m High ACK times[2], to Low if the insertion is successful (i.e., there is space in the CB). As stated we configure the Pump as a high process to allow communication between the Pump and High. However, as discussed, the sending of ACKs to Low violates the Bell-LaPadula constraints. A Trojan horse can exploit this procedure. If the message arrives while the CB is empty, then the TLP will send the signal, *wake-up*, to the THP.

High: (Exterior to the Pump)

When High receives a message from the THP, it stores the message and then sends an ACK/NAK to the THP.

[1]Note that sliding window based schemes exist that can send w messages without receiving any ACK/NAK. For simplicity, we assume $w = 1$ throughout the discussion in this paper.

[2]The choice of the delay is the key to a successful implementation of the Pump, see section 3.4.

Trusted high process:

Upon receiving the *wake-up* signal from the TLP, it repeats the following while there is a message in the CB:

1. Send the first message in the CB to High.
2. Once an ACK arrives from High, remove the message from the CB and compute the moving average. If the THP receives NAK or no response (i.e., time-out) then retransmit the same message.

Since the Pump is configured as a high process, this communication between the THP and the high process does not violate the Bell-LaPadula constraints.

Communication Buffer:

This is a regular FIFO buffer whose length is n, which the TLP and the THP share. The TLP and the THP may also learn certain statistical information from the CB (e.g., how many messages are pending in the buffer, etc.).

We will use the notation $P(G, m, n)$ to signify a Pump with a communication buffer of size n, designed with a delay given by the random variable G based on a moving average of the last m High ACK times. In section 3.4 we use a modified exponential random variable and this specific Pump is denoted by $P(M, m, n)$, where we use M to represent the exponential (memoryless) random variable.

It is easy to see that any process that communicates with the Pump can collect garbage because this process receives ACKs. Also, any communication with the Pump is reliable due to ACK/NAK being sent. If the sender receives either NAK or is timed out, then it will retransmit the same message. This communication method is also recoverable if we implement the CB in non-volatile storage and each message has an associated message number. We consider the following four cases:

case 1: *The system crashes after Low sends a message to the Pump but before the Pump receives it.* Since Low never receives an ACK, it will resend the message as the system recovers.

case 2: *The system crashes after the Pump receives a message but before Low receives an ACK.* Since Low never receives an ACK, it will resend the message as the system recovers. However, the Pump will notice that the message has already been received because of the message number. Hence, it will just send an ACK and ignore the message.

case 3: *The system crashes after the Pump sends a message but before High receives it.* This is similar to case 1.

case 4: *The system crashes after High receives a message but before the Pump receives an ACK.* This is similar to case 2.

Note that there may be many destination entities and many source entities that use the same Pump. However, in this paper, we just consider the case of one source entity and one

destination entity which will have the worst case covert channel capacity (less noise, see [4] for the application of the Pump in networks where many Lows send messages to many different Highs). We have been denoting these two processes of interest as simply High and Low. Further, when we perform channel capacity analysis we assume that there are no NAKs to Low. This makes the analysis easier but does not affect the capacity bounds.

3.3 Performance/Security goals of the Pump

The random variable that delays the Low ACK is chosen so that R_l is roughly equal to R_h. Note that if the arrival rate is slower than R_h, slowing R_l down to the R_h does not affect throughput because R_l is still faster than the arrival rate. We discussed the security reasons for this in subsection 3.3.1 and we will discuss an actual realization of the random variable in section 3.4.

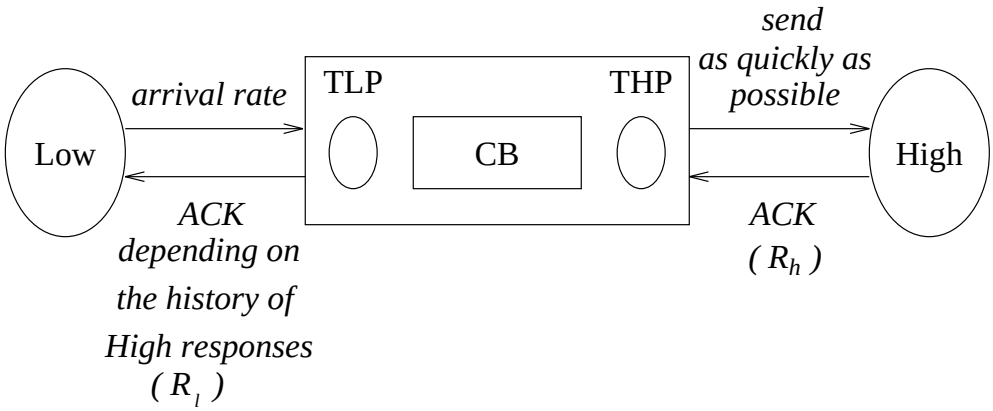

Figure 7: Rates associated with the Pump.

At first glance it seems that slowing Low down will adversely affect the performance of our communication throughput. In section 5, we show that for the $P(M, m, n)$, discussed in section 3.4, that this is not the case when we compare performance of the Pump to that of the conventional communication protocol through simulations.

3.3.1 Low ACK rate = High ACK rate

We guarantee that R_l is roughly equal to R_h via the moving average construction in the TLP. Let L_i be the random variable that represents the time it takes for Low to receive an ACK from its ith message sent into the CB. Let H_{m_i} be the average of the last m High ACK times that have occurred up to and including the time that the CB receives the ith message from Low.

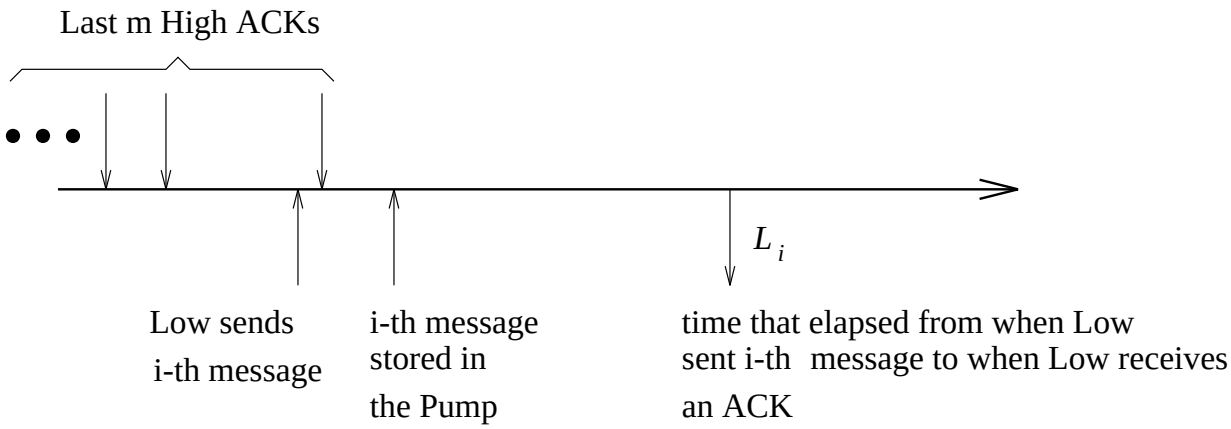

Figure 8: Moving average construction.

A necessary requirement is that the mean of L_i, denoted as $\mu(L_i)$ be equal to H_{m_i}. Even though the L_i are not independent (or identical to each other) we take a "strong law of large numbers" approach toward looking at R_l. Note that R_l is the inverse of the average of the actual values of the L_i. This numerical average should behave as the average of the $\mu(L_i)$ but this, by definition, is equal to the average of the High moving averages the H_{m_i} and this is approximately equal to the average of the High ACK times (provided that the total number of messages acknowledged by High, denoted by N, satisfies $N >> m$) which is simply $1/R_h$. Thus,

$$\frac{1}{R_l} = \frac{1}{N}\sum L_i \approx \frac{1}{N}\sum \mu(L_i) = \frac{1}{N}\sum H_{m_i} \approx \frac{1}{R_h} \ .$$

Hence $R_l \approx R_h$. Note it is not our goal to show that the Pump achieves $R_l \approx R_h$; rather it is our goal to show that the Pump has a built-in mechanism that does not allow the CB to become full too often and, when it is full, that substantial information is not passed to Low. Therefore, we use the above mathematics to give justification to our choices for the random variables defining the Low ACK time. Our analysis of covert channel exploitations in section 4 shows that Low cannot glean significant information if and when the CB becomes full.

3.3.2 Throughput

Since the Pump is a one-way communication device rather than a two-way communication device (that can be used in an interactive mode), the *throughput* is a good measure of performance. The throughput is the number of messages sent from Low to High, that are acknowledged to the buffer by High, per unit time.

When studying performance, we are considering only the benign case. When a Trojan horse is acting maliciously, performance is no longer our first priority. One of the advantages of the Pump is that its performance to security ratio is dynamic. By this we mean that in the benign case the performance takes a very small hit; however, in the Trojan horse case the capacity of any potential covert channels is greatly diminished and that the Pump responds automatically to the changing situations.

To ensure good performance/security we believe that the buffer should not be allowed to become and/or stay full. We have already discussed the security problems relating to a full buffer. The performance reasons are that, if Low is sending messages faster than High can handle them, then High becomes a bottleneck and the throughput is thus limited by the High rate.

However, the construction of the Pump has the ACK times to Low governed by a random variable by definition.

1. If High is faster than Low then the random variable should not affect Low's rate because R_l is faster than Low's rate.
2. If Low is faster than High, then the random variable should slow Low's rate down. This is the reason for the moving average construction in the Low ACK time. In terms of performance the best that we can hope for is Low's rate to be equal to High's rate. If Low and High rates are roughly equal then performance and security are good. If Low is allowed to become slower than High then we are wasting resources since High waits in an idle mode for Low to send messages.
3. If Low and High rates are equal then we are all right since the probability of the buffer becoming and staying full are small (burstiness may occur).
4. A fourth case arises if the source of delays is the Pump itself (i.e., the overhead). In this case the buffer can become full but it is not under the control of High. Even though this is acceptable for security it is not good for performance and a faster Pump should be used. We will not consider this case further.

3.4 Choice of a random variable — the $P(M, m, n)$

The density function of the random variable for Low ACK times that will be chosen should have the following two properties:

1. *The mean of this random variable should be controllable.* The density function should be sensitive to system feedback, in order to meet the performance/security requirement.
2. *There should be no upper bound.* If the support of the density function has an upper bound, then the upper bound can be exploited by Trojan horses. For example, if the uniform distribution is chosen with support $[O_v, B]$ the mean must be H_{m_i} for the performance reasons discussed. This forces $B = 2H_{m_i} - O_v$. Hence, if the high Trojan horse decides to send a signal by keeping the CB full for a time greater than B, then the signal is delivered without any noise. Aside from capacity concerns this allows small messages to be passed, with high confidence, in a small amount of time.

Even though there are many random variables that satisfy the above properties we have chosen the exponential distribution because the capacity of the covert channel is relatively easy to analyze (due to the relatively simple density function).

We will now describe in detail an implementation of the Pump using a modified exponential distribution for the delay in the ACKs to Low. This is the $P(M, m, n)$. As before, we let O_v be the communication overhead for the Pump. By this we mean that O_v is the minimum value for any L_i. S_i is the value for what L_i would be if this were just the SAFP. The smallest S_i can be is O_v (which is always the case if the buffer is not full). The largest S_i can be is τ, where τ is a time out which is taken to be much larger than O_v. The ith response to Low, L_i, is given by a random variable that has the density function $f_i(t)$. There are three cases to discuss.

Case 1: $S_i = O_v$ and $S_i < H_{m_i}$

This is always the situation when the CB is not full.

$$f_i(t) = \begin{cases} \alpha_i e^{-\alpha_i(t-S_i)}, & if \;\; O_v \leq t < \tau, \\ 0, & otherwise. \end{cases}$$

This is just an exponential distribution that starts at S_i instead of time 0. Unfortunately, this will allow the response time to be infinite, so we must bound this distribution. We therefore put a time out in such that if Low has not received an ACK by time τ, it interprets that as a NAK. We should then adjust the density function by a multiplicative constant. However, τ is chosen so much larger than O_v, which is the present S_i value, that this constant is essentially unity and we therefore ignore the constant.

The mean of the above density function is $S_i + 1/\alpha_i$. Since we wish for this mean to be equal to the moving average of the last m High ACK times we see that $\alpha_i = \frac{1}{H_{m_i} - S_i}$.

Case 2: $S_i > O_v$ and $S_i < H_{m_i}$

As above $\alpha_i = \frac{1}{H_{m_i} - S_i}$. The timeout τ is much greater than O_v, but there is no guarantee that it is much larger than a generic S_i value. Therefore if S_i is much larger than O_v we can no longer assume that the tail of the modified exponential distribution is negligible. Therefore, we cannot ignore the above mentioned multiplicative constant and we must modify what we did in case 1. However, the spirit is still the same.

Step 1: Set $\beta_i = \tau - 1/\alpha_i$. We still use an exponential distribution between S_i and β_i, however, we "absorb" all of the probability (assuming exponential behavior) from β_i to ∞ into time τ. To be precise, consider the following pdf:

$$g_i(t) = \chi_{[S_i, \beta_i]}(t) \alpha_i e^{-\alpha_i(t-S_i)} + e^{-\alpha_i(\beta_i - S_i)} \delta(t - \tau)$$

Note, since $\beta_i - S_i = \tau - H_{m_i} \geq 0$ and $\delta(t)$ is the Dirac delta function, the above is a well-defined density function. The expectation of this density is

$$\int_{-\infty}^{\infty} t g_i(t)\, dt = S_i + \frac{1}{\alpha_i} + e^{-\alpha_i(\beta_i - S_i)}(\tau - \beta_i - \frac{1}{\alpha_i})$$

Since β_i was set equal to $\tau - 1/\alpha_i$ we see that the expected value is simply $S_i + 1/\alpha_i$. We could stop at this step, however there is now a non-zero probability of $L_i = \tau$. This could possibly be used for covert communication so we wish to further increase the ambiguity of L_i while still keeping performance in mind.

Step 2: Now we select a uniform random number between S_i and β_i and call it β'_i and set $\tau'_i = \beta'_i + 1/\alpha_i$. Consider yet another pdf

$$g'_i(t) = \chi_{[S_i,\beta'_i]}(t)\alpha_i e^{-\alpha_i(t-S_i)} + e^{-\alpha_i(\beta'_i - S_i)}\delta(t-\tau'_i)$$

Since this integrates over the reals to one it is a well-defined density function. Its expected value is still $S_i + 1/\alpha_i$. So we have negated using τ as a signal. Since τ'_i is in fact randomly generated, we feel there is a very small chance that τ'_i may be used as a signal. However, we go one step further to increase the ambiguity of the time signal.

Step 3: We wish to uniformly spread the probability of $L_i = \tau'_i$ out over the interval between $[\tau'_i, \tau]$. Consider the density function

$$f_i(t) = \chi_{[S_i,\beta'_i]}(t)\alpha_i e^{-\alpha_i(t-S_i)} + \chi_{[\tau'_i,\tau]}(t)\frac{e^{-\alpha_i(\beta'_i - S_i)}}{\tau - \tau'_i}$$

Now the mean of this is larger than $S_i + 1/\alpha_i$. This is acceptable because this helps keep the CB from remaining full. This does not affect performance too much because the percentage of time that the CB is full is in fact quite small (see section 5).

Case 3: $S_i \geq H_{m_i}$

This is identical to case one, except we use a small mean for the exponential distribution. For the best performance the mean should be infinitesimally small, however for security we bound it away from zero, otherwise the value of S_i can be observed by Low (and possibly be used for noiseless covert communication).

3.5 Pump algorithm

We give the algorithm for the $P(M, m, n)$ to show how the Pump sets L_i.

Note: S_i is the time between when Low sends a message and when Low receives an ACK in the SAFP and ϵ is a small number. We have the following relationships between our terms. $\beta_i = \tau + S_i - H_{m_i}$, $\tau_i{}' = \beta_i{}' + H_{m_i} - S_i$, $\beta_i{}'' = \beta_i{}' - S_i$ and a random delay $D_i = L_i - S_i$.

Message is placed in CB
Read H_{m_i} ;
IF $S_i \geq H_{m_i}$ THEN $\mu := \epsilon$;
ELSE $\mu := H_{m_i} - S_i$;
END IF ;
Draw an exponential random number ξ whose mean is μ ;
IF $S_i = O_v$ OR $S_i \geq H_{m_i}$ THEN
    $D_i := \xi$;
    IF $D_i > \tau - S_i$ THEN $D_i := \tau - S_i$;
    END IF ;
ELSE

Draw a uniform random number β_i'' between 0 and $\beta_i - S_i$;
IF $\xi \leq \beta_i''$ THEN $D_i := \xi$;
ELSE
Draw an uniform random number ξ' between τ_i' and τ ;
$D_i := \xi'$ - S_i ;
END IF ;
END IF ;

4 Covert Channel Analysis

We will now show that it is impossible for a Trojan horse to exploit the Pump in any meaningful way. Assume that High wishes to signal Low covertly. Let us try to get some quantitative bounds on the capacity for a $P(M, m, n)$.

High will attempt to signal Low by affecting the values of L_i. Say High tries the strategy that we discussed earlier of letting the CB get full and then removing messages within time $O_v + i\epsilon$. A few factors make this an unfeasible Trojan horse strategy. High cannot get the CB full and keep it full without imposing a severe time penalty being enacted upon L_i. This is because for the CB to become full, High must be removing messages at a slower rate than Low is getting ACKs back from the TLP. But after a certain number of messages the slow rate of High is manifested by forcing L_i to also slow down due to the moving average construction of $\mu(L_i)$. There are three basic problems with this approach.

- The noise that is involved when High tries to send a symbol to Low.
- The time involved in sending the symbol due to large delays by High necessitated by High trying to send a symbol with as little noise as possible.
- Synchronization problems between High and Low. By this we mean the ability of Low to differentiate, via L_i values or the number of messages ACK'ed, between when High is getting ready to send a message (i.e., letting the CB get full) and when L_i is the actual symbol being passed by High.

Let us consider three possible exploitations below where the last strategy exploits High's ability to influence H_{m_i} and the first two strategies exploit High's ability to make CB full in addition to influence H_{m_i}.

Exploitation strategy 1:

1. High acts quickly (ACK time = O_v) m times. This has the effect of lowering the moving average and thus speeding up the L_i values.
2. Now High does not send an ACK for $t = nO_v$ in the hopes of Low filling the CB. When Low finally does receive this delayed L_i value it is interpreted as a synchronization signal

from High to Low — This means that the next L_i value is to be interpreted as a symbol being sent by High.

3. High sends a symbol through the High ACK time (i.e., via the S_i value chosen by High). Note that due to the probabilistic nature of L_i and the fact that the CB may not even be full, this symbol is quite ambiguous (noisy). Also now L_i is large because of the previous High delay of $t = nO_v$.

High wishes for the CB to become full again so that it can again send a symbol with as little noise as possible, so High repeats the above process of lowering L_i by acting quickly and then delaying and finally sending a symbol. We see that if a symbol is sent noiselessly it would take at least $t = (m+n+1)O_v$. The next shortest symbol would take $(m+n+1)O_v + \epsilon$, the next $(m+n+1)O_v + 2\epsilon$, etc. We assume (the worst case) that the channel has an infinite alphabet, so, like before, the capacity of this timing channel (C') is the logarithm of the positive root of $1 - (x^{-(m+n+1)O_v} + x^{-\epsilon})$. Obviously this timing channel has a smaller capacity than the FBC (we designate the capacity of the FBC by C for this discussion). However, the question remains as to how much smaller. We define R by

$$R = \frac{-N}{q} \frac{\log(1 - w^{-1/N})}{\log w}$$

where w is the positive root of $1 - (w^{-q} + w^{-1})$. If $m + n + 1 \geq R$, it can be shown that $C' \leq \frac{1}{N} C$. Thus by adjusting m and n we can reduce the capacity by any amount that we want. Follows a plot of R against the reduction factor $\frac{1}{N}$ for various q values.

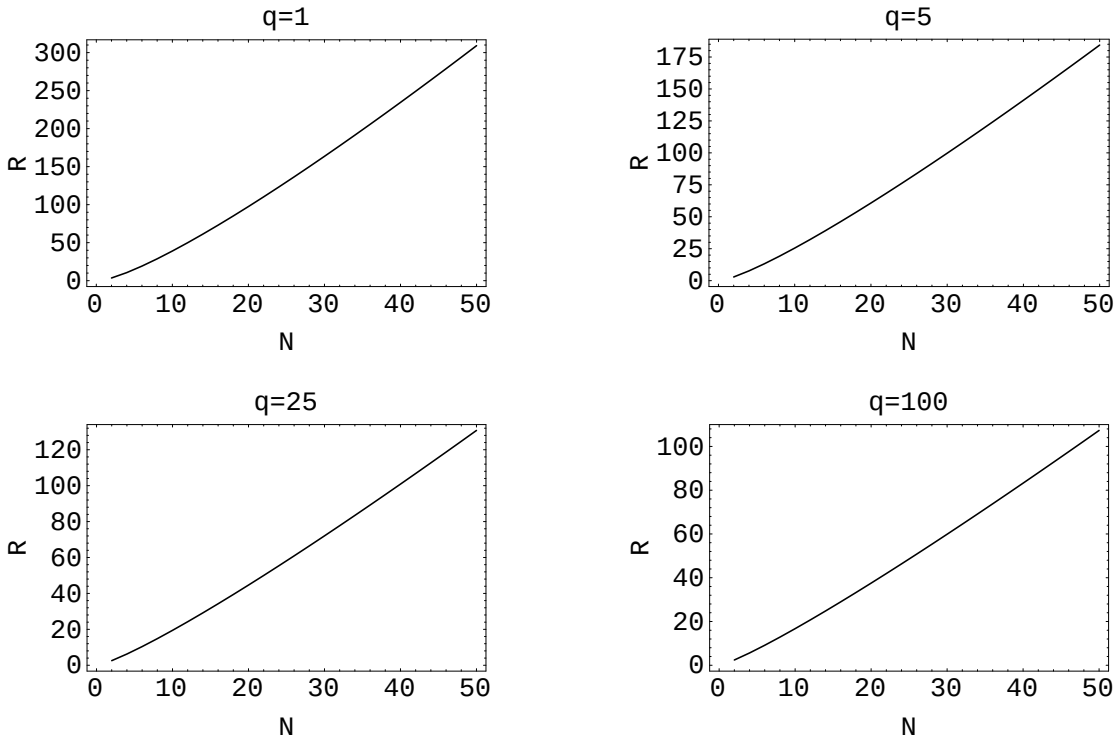

Figure 8.a: reduction factors.

Exploitation strategy 2:

1. High acts quickly (ACK time = O_v) m times.
2. Now High does not send an ACK for $t = nO_v$.
3. High sends a symbol through High ACK time (i.e., via the S_i value chosen by High).

The first three steps are the same as those of the strategy 1. The difference is instead of High repeating the process of — filling the CB, delaying, sending a symbol, and filling the CB again — after High sends the first symbol, it continues to send symbols. However, if High ACKs a message quickly (i.e., small S_i values) to try to send more symbols per given time to Low it will, in fact, end up only emptying out the CB and thus will not be able to send Low different S_i values. High does not know when Low sends a message to the Pump; therefore, High must be careful (if it wishes to keep the CB full) not to ACK a message too quickly from the THP, because if it ACKs too quickly the CB will no longer be full and High cannot send symbols by using different S_i values. Unless the High ACK times are essentially the timeout

τ, the probability the CB is full approaches 0 as the number of symbols being sent increase.[3] However, the closer the High ACK times are to τ, the less High can manipulate the S_i values; therefore, the number of symbols that High can send decreases as the probability of keeping the CB full increases. With all of this in mind, we feel that this exploitation strategy cannot have a capacity of more than $1/\tau$ bits per tick.

Exploitation strategy 3:

High could attempt to send information to Low by simply affecting the moving average and having Low interpret its response times without High trying to make the CB full. A full analysis of this scenario is quite complicated and involves channels with continuous outputs. Also, there are severe practical coding issues when one quantizes the output space into many symbols. So even though a true capacity upper bound could be obtained, it would be quite difficult to build the proper code. From a practical standpoint one could study the capacity just through finite decoding schemes (this is not to say that one should not see how the capacities differ). We can make some qualitative statements about the channel capacity based on present techniques. Low's ACK time is a modified exponential distribution with shift O_v. All that High can do is to alter the mean. Let us look at a simplifying example where High tries to send a symbol to Low by varying the mean between two values, keeping in mind that any immediate effect High could have on the mean is moderated by the moving average construction. Say Low receives a response and wants to decide whether it came from a modified exponential distribution with mean 1 or mean 2. If the means are close then it is hard to make this decision and the symbol is very noisy. To make the symbol less noisy would require High to enlarge the difference between the means; this, however, would also increase the time that Low receives the symbol and in fact increase the time that Low receives future symbols due to the moving average construction of the means. Therefore, we decrease the noise with which symbols are sent only by penalizing the time cost with which they are sent. Between the fidelity criterion of the symbols forcing a large difference between the values of the means and the fact that the moving average moderates any change of High by a factor of $1/m$, we feel that the capacity of these exploitation scheme is $1/m$ that of the FBC. We realize that we have only given an intuitive argument for this $1/m$ reduction. At present, we are investigating more precise arguments for this reduction [17].

Certainly one could use a combination of the above exploitation strategies. However, we do not see any order of magnitude improvement by doing this. In our capacity bounds for all three exploitations we were tacitly assuming that the Low and High overheads are equal. This only affects exploitation strategy 1. If the overheads differ, the exponents of the polynomials differ, but by adjusting m and n we can still get the desired capacity reduction.

A modification could be done to the $P(M, m, n)$ so that the mean of the exponential

[3] $P($ CB full after sending k symbols $) = \prod_{i=1}^{k} P($CB full after sending symbol $i)$, since $P($CB full after sending symbol $i)$ is bounded away from 1, provided that the High ACK times are bounded away from τ we see that this product approaches 0, relatively quickly as k increases.

distribution was a *function* of H_{m_i}. However, if one wishes to reduce the channel capacity further, λ can be chosen not only as a function of H_{m_i} but also as a function of the current state of the CB. For example, if the CB is 80% full then α_i may be a function of $2H_{m_i}$, if the CB is 90% full then α_i may be a function of $3H_{m_i}$, and so on. This will have the effect of slowing down L_i when High tries to send covert signals with very little noise.

5 Simulation Analysis

To substantiate our performance claim, we perform a simulation. First, we describe our input source, server, Pump and SAFP models. We then show the throughput rates and the profile of the CB.

5.1 Simulation Model

In this section we present a performance comparison of the Pump to the conventional communication protocol via a simulation model. We make the standard assumptions regarding the (benign) usage of either protocol—namely, that they are single-server queues with exponential interarrival times and a service times given by the 2-Erlang distribution. In other words a M/G/1 queue. Often queues such as these are modeled by the simpler M/M/1 queue. The M/M/1 queue is more tractable for closed form analysis but the M/G/1 queue is a better representation of reality [18]. Since our interest lies in simulation, not analytic, results, we will use the more realistic model of queue behaviour.

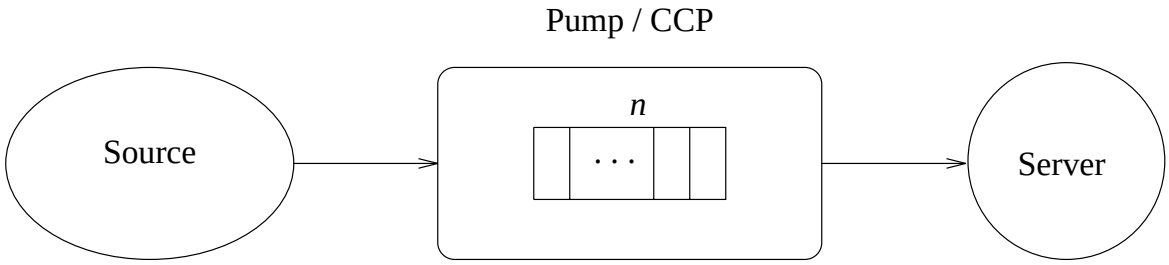

Figure 9: Simulation model.

The Pump and the SAFP use the same simulation model except moving average. When the SAFP is simulated, the moving average is always set to zero. One the other hand, when the Pump is simulated, the moving average is computed depending on the server's ACK time. We assume that all messages have the same length, the timeout, τ, is 250.0 ms, the overheads, O_v, of store and forward buffer and the CB buffer to process a message are both 0.3 ms, and a small number, ϵ, is 0.001 ms in our experiments. We ignore all factors that are common in both the SAFP and the Pump, and try to isolate the effect of random ACK time to throughput.

We perform three classes of experiments as follows. The source attempts to generate messages whose interarrival mean time is 1.0 ms (the action of the Pump and Low's internal

finite buffer size of course moderate this generation). The server processes messages according to a 2-Erlang distribution with mean service time 0.5 ms, 1ms, or 2ms, depending on the experiment. These experiments correspond to Low being slower, the same, or faster, than High, respectively. Each class of experiment is broken down into three subclasses of different buffer and moving average sizes.

5.2 Simulation Results

All simulations have been run ten times with different random numbers. The following results are the average of those runs and show three aspects: (1) throughput, (2) the average length of the queue in the Pump and the SAFP, and (3) the probability of the buffer to be full (CBf).

Service Time = 0.5 ms, High faster

Table 1. n = 10, m = 10

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf (%)
SAFP	1001	1.8	0.0
Pump	1002	1.6	0.0

Table 2. n = 100, m = 100

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf (%)
SAFP	1001	1.8	0.0
Pump	1001	1.6	0.0

Table 3. n = 1000, m = 1000

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf (%)
SAFP	1001	1.8	0.0
Pump	1003	1.6	0.0

Service Time = 1.0 ms, High same as Low

Table 4. n = 10, m = 10

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf(%)
SAFP	992	9.7	81.6
Pump	960	7.6	20.6

Table 5. n = 100, m = 100

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf(%)
SAFP	992	75.5	49.3
Pump	989	32.1	0.0

Table 6. n = 1000, m = 1000

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf(%)
SAFP	993	149.1	0.0
Pump	990	46.0	0.0

Service Time = 2.0 ms, High slower

Table 7. n = 10, m = 10

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf(%)
SAFP	500	10.0	96.2
Pump	475	7.6	24.5

Table 8. n = 100, m = 100

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf(%)
SAFP	500	99.9	95.8
Pump	496	48.9	0.0

Table 9. n = 1000, m = 1000

	Throughput	Queue	
	(messages/sec)	Mean Length	CBf(%)
SAFP	500	987.0	91.2
Pump	494	82.1	0.0

The simulation results show that (1) there is little performance penalty due to randomized ACKs from the Pump and (2) the Pump substantially lower the CBf(%) (that in turn lowers the covert channel capacity). Therefore we see that the Pump does not hurt performance, but does prevent the CB from becoming/staying full.

6 Summary

A Pump that balances the communication requirements of reliability and security is introduced. The Pump provides ACKs to a source for reliability. These ACKs are also used to regulate the input rate from a source by attempting to slave the input rate to the inverse of the moving average of the service time through the randomized ACK rate to the source.

Despite the Pump's randomized ACKs, there still exists a covert channel. We analyzed the capacity of this covert channel as a function of buffer size and moving average size. The purpose of the covert channel analysis is to provide guidelines for the designer of the Pump to choose appropriate design parameters (e.g., size of buffer and moving average) depending on the analysis presented in this paper and system requirements.

To back up our performance claim in this paper, we presented simulation results.

7 Acknowledgements

We would like to thank Ruth Heilizer, Carl Landwehr, and Ravi Sandhu for their helpful comments.

References

[1] D. Denning. "The lattice model of secure information flow," Communications of the ACM, Vol. 19, No. 5, 1976.

[2] R.S. Sandhu. "Lattice-based access control models," Computer (IEEE), Vol. 26, No. 11, pp. 9-19, Nov. 1993.

[3] D. Bell and L. LaPadula. "Secure computer systems: Mathematical Foundation," ESD-TR-73-278, Vol.1, Mitre Corp, 1973.

[4] M.H. Kang, Ira S. Moskowitz and D. Lee."A network version of the Pump," Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1995.

[5] M. Schwartz. Computer Communication Network Design and Analysis. Prentice Hall, 1977.

[6] J. McDermott. "The b2/c3 problem: how big buffers overcome covert channel cynicism in trusted database systems," Proc. of the IFIP WG 11.3 eighth annual working conference on database security, Germany, August 1994.

[7] Ira S. Moskowitz and A.R. Miller. "The channel capacity of a certain noisy timing channel," IEEE Transactions on Information Theory, Vol. 38, No. 4, pp. 1339-1344, July 1992.

[8] C. Shannon and W. Weaver. The mathematical theory of communication. University of Illinois Press, 1949. Also appeared as a series of papers by Shannon in the Bell System

Technical Journal, July 1948, October 1948 (A Mathematical Theory of Communication), January 1949 (Communication in the Presence of Noise).

[9] Ira S. Moskowitz and M. H. Kang. "Covert channels — Here to stay?," Proceedings of COMPASS '94, pp. 235 - 243, Gaithersburgs, MD, 1994.

[10] B. Lampson. "A note on the confinement problem," Communications of the ACM, Vol. 16, No. 10, 1973.

[11] National Computer Security Center. DoD Trusted Computer Security Evaluation Criteria (Orange Book). DoD 5200.28-STD, 1985.

[12] O. Costich and Ira S. Moskowitz. "Analysis of a storage channel in the two-phase commit protocol," Proc. of the Computer Security Foundations Workshop 4, Franconia, NH, 1991.

[13] J. W. Gray III. "On introducing noise into the bus-contention channel," Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, 1993.

[14] W.M. Hu. "Reducing timing channels with fuzzy time," Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, 1991.

[15] A.R. Miller and Ira S. Moskowitz. "Reductions of a class of Fox-Wright Psi functions for certain rational parameters," Computers & Mathematics with Applications, to appear.

[16] Ira S. Moskowitz and A.R. Miller. "Simple Timing Channels," Proc. of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, 1994.

[17] Ira. S. Moskowitz and M. H. Kang. "Discussion of a statistical channel," Proceedings of IEEE-IMS Workshop on Information Theory and Statistics, p. 95, Alexandria, VA, 1994.

[18] A.M. Law and W.D. Kelton. Simulation Modeling and Analysis. McGraw Hill, (1982) 1991.